**Name : Akash Babu Misal**

**PRN: 21520005**

**Class:** Final Year (Computer Science and Engineering)

**Course Name: Cryptography and Network Security**  **Lab**

**Assignment No – 10**

**Aim**: Diffi-helman key exchange Algorithm

**Theory**:

**Diffie-Hellman Key Exchange Overview:**

The Diffie-Hellman key exchange is a public-key cryptosystem that allows two parties to securely exchange cryptographic keys over an untrusted communication channel. This method provides a way for parties to agree on a shared secret key without directly transmitting that key. It is fundamental to modern cryptography and used in many secure communication protocols, including TLS (Transport Layer Security) and SSH (Secure Shell).

**Algorithm Steps:**

1. **Public Parameters**: The algorithm relies on publicly known parameters:

* A large prime number 'p.'
* A primitive root 'g' modulo 'p,' which is a number whose powers cover all possible remainders when divided by 'p.'

1. **Key Generation:**

* Each party, Alice and Bob, chooses a private key: 'a' for Alice and 'b' for Bob. These keys are kept secret.

1. **Partial Exchanges**:

* Alice computes 'A = g^a mod p' and sends 'A' to Bob.
* Bob computes 'B = g^b mod p' and sends 'B' to Alice.

1. **Shared Secret:**

* Both Alice and Bob can now compute the shared secret key:
  + - * Alice: 's = B^a mod p'
      * Bob: 's = A^b mod p'
  + The resulting shared secret 's' is the same for both Alice and Bob.

**Security of Diffie-Hellman:**

The security of the Diffie-Hellman key exchange relies on the difficulty of the discrete logarithm problem. Given 'p,' 'g,' and 'A,' it should be computationally infeasible for an eavesdropper to determine 'a.' In other words, finding 'a' from 'p,' 'g,' and 'A' should be a challenging task, especially when 'p' is large enough.

**Code:**

#include <bits/stdc++.h>

using namespace std;

void file()

{

#ifndef ONLINE\_JUDGE

    freopen("input.txt", "r", stdin);

    freopen("output.txt", "w", stdout);

#endif

}

long long powM(long long a, long long b, long long n)

{

    if (b == 1)

        return a % n;

    long long x = powM(a, b / 2, n);

    x = (x \* x) % n;

    if (b % 2)

        x = (x \* a) % n;

    return x;

}

bool checkPrimitiveRoot(long long alpha, long long q)

{

    map<long long, int> m;

    for (long long i = 1; i < q; i++)

    {

        long long x = powM(alpha, i, q);

        // cout << x << endl;

        if (m.find(x) != m.end())

            return 0;

        m[x] = 1;

    }

    return 1;

}

int main()

{

    file();

    long long q, alpha;

    q = 71;    // A prime number q is taken

    alpha = 7; // A primitive root of q

    if (checkPrimitiveRoot(alpha, q) == 0)

    {

        cout << "alpha is not primitive root of q";

        return 0;

    }

    else

    {

        cout << alpha << " is private root of " << q << endl;

    }

    long long xa, ya;

    xa = 4;                  // xa is the chosen private key

    ya = powM(alpha, xa, q); // public key of alice

    cout << "private key of alice is " << xa << endl;

    cout << "public key of alice is " << ya << endl

         << endl;

    long long xb, yb;

    xb = 3;                  // xb is the chosen private key

    yb = powM(alpha, xb, q); // public key of bob

    cout << "private key of bob is " << xb << endl;

    cout << "public key of bob is " << yb << endl

         << endl;

    // key generation

    long long k1, k2;

    k1 = powM(yb, xa, q); // Secret key for Alice

    k2 = powM(ya, xb, q); // Secret key for Bob

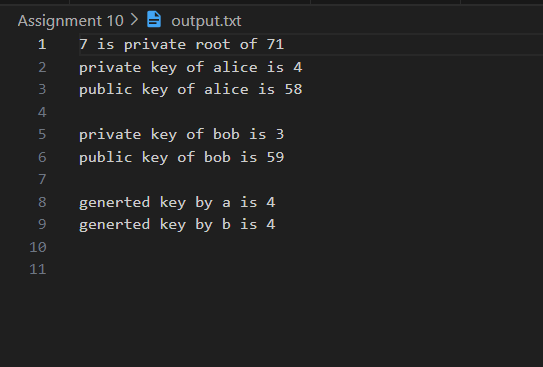
    cout << "generted key by a is " << k1 << endl;

    cout << "generted key by b is " << k2 << endl

         << endl;

    return 0;

}

****

**Conclusion:**

The Diffie-Hellman key exchange algorithm is a fundamental concept in modern cryptography that enables secure key exchange over untrusted communication channels. This experiment allows us to understand the principles of the Diffie-Hellman algorithm and its use in securely establishing a shared secret key between two parties. It emphasizes the importance of discrete logarithm hardness for the security of this method and its widespread application in secure communication protocols.